A Multi-objective Evolutionary Algorithm for Rule-based Performance Optimization at Software Architecture Level

Xin Du
Faculty of software
Fujian Normal University
Fuzhou, China
School of Computer Science
The University of Birmingham
Birmingham, UK
xindu79@126.com

Youcong Ni
Faculty of software
Fujian Normal University
Fuzhou, China
youcongni@foxmail.com

Peng Ye
College of Mathematics and Computer
Wuhan Textile University
Wuhan, China
yepeng@wtu.edu.cn

1. INTRODUCTION

Architecture-based software performance optimization can not only significantly save time but also reduce cost. At present, researchers have proposed rule-based and metaheuristic-based approaches. Metaheuristic-based approaches can only explore a few of architectural parameters, such as component allocation, hardware configuration, and component selection. Moreover, the majorities of metaheuristic-based approaches do not consider how to apply architecture-based software performance improvement knowledge in the evolutionary optimization process. However, the existing rule-based approaches may only explore a limited region of the improvement space because they do not fully take the uncertainty of the count and order of each rule usage into account. As a result, the search space for performance improvement is limited so that the optimal solution is hard to find out. Aiming to the problem, we have proposed an evolutionary algorithm for rule-based performance optimization at software architecture (*SA*) level named EA4PO. In EA4PO, response time and the count of rules usage with improvement effect have been combined into a single objective by means of weighting. But there are two drawbacks in EA4PO. One is that the value of weight is determined by experiments. Another is that EA4PO can just obtain one optimal solution. In fact, software architect wants a set of trade-off candidates more. In order to improve the performance of EA4PO algorithm, a multi-objective evolutionary algorithm for rule-based performance optimization at *SA* level named MOEA4PO is proposed in this paper. In MOEA4PO, response time and the count of rule usage with improvement effect are considered as two objectives. An individual is encoded as a sequence of rule number. An adaptive mutation operator is designed to guide the search direction by fully considering heuristic information of rule usage during the evolution. A well-known non-dominated sorting genetic algorithm II (NSGA II) is chosen as the main framework. Case study shows that MOEA4PO can obtain better results than EA4PO [1] and a typical rule-based approach [2].

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems-modelling techniques; D.2.8 [**Software Engineering]:** Metrics-performance measure; D.2.11 [**Software Engineering]:** Software Architecture

Keywords

Performance analysis; performance optimization algorithm; evolutionary algorithm; software architecture

2. BACKGROUND

Rule Sequence Execution Framework (RSEF) we proposed [1] can support the execution of rule sequence. In RSEF, the initial *SA* and the sequence of rule number are inputs. The outputs of RSEF are the response time of system before and after the execution of rule sequence, and the information table of rule usage in sequence, which can be used to compute the count of rule usage with improvement effect in a sequence.

3. MOEA4PO ALGORITHM

3.1 Individual encoding

The performance improvement rules are sequentially numbered from 1 to *n,* and each rule has its maximal usage times. The rule numbered by 0 is introduced to represent the do-nothing rule and has no effect on the performance improvement of system. The maximal usage times of rule 0 are equal to the sum of maximal usage times of the rules numbered by non-zero.

An individual *X* is encoded as a sequence of rule number with fixed-length integer. The length of *X* is equal to the maximal usage times of the rule numbered by 0. The rule number in *X* has two constraints. One is that the value of each gene in *X* varies from 0 to *n*. Another is that the actual occurrence times of each rule number in *X* should be less than or equal to maximal usage times of corresponding rule.

The aim of introducing rule number 0 is to ensure that each rule number in *X* can comply with the two constraints mentioned above and get the shorter sequence of rule numbers.

3.2 The objective functions

There are two objective functions. One is to obtain the final response time of the system based on individual *X*, which can be got directly by RSEF. Another is to compute the count of rules usage with improvement effect in individual *X*, which is computed using the information table of rule usage in sequence obtained by RSEF. The count of rules usage with improvement effect can indirectly represent the cumulative cost during the optimization, which is just to allow cost to be considered. The optimization objective is to minimize the response time of system and the count of rules usage with improvement effect.

3.3 Crossover operator

One-point crossover with constraint checking and repairing is adopted. And it includes three computational steps: crossover, constraint checking and repairing. First, two intermediate individuals are generated by crossover operator. Then, the constraint checking is done on two intermediate individuals to verify whether each gene from the crossover position to the last position satisfies the constraint defined in section 3.1. Finally, repairing step will be done for those genes which disobey the constraint and assign 0 to them.

3.4 Adaptive mutation operator

The mutation operator has a great influence on the convergence rate of evolutionary algorithm. Here, the adaptive mutation operator with learning tactics is introduced to guide the search direction of algorithm. Each gene mutates according to the conditional mutation probabilities $p(x_j = k \mid x_{j-1} = q)$ that are computed from the statistical information of rule usage during the evolution, where *j* is the mutation position, *k* and *q* are rule number at *j*th and (*j*-1)th positions.

The mutation operator in MOEA4PO algorithm is also composed of three computational steps: mutate, constraint checking and repairing. The constraint checking and repairing are same as crossover operator.

4. CASE STUDY

In order to validate the effectiveness of MOEA4PO algorithm, web application (WebApp) [2] is selected as the experimental case to compare MOEA4PO algorithm with EA4PO algorithm [1] and Xu's DFS algorithm [2] . The EA4PO and DFS algorithms are applied to the WebApp case for obtaining the shortest response time. And corresponding parameters setting are given in [1]. The response time and the count of rule usage with improvement effect obtained by DFS are 29.88 ms and 8 respectively. The average response time and the count of rule usage with improvement effect obtained by EA4PO algorithm are 26.50ms and 7. Experimental results show that EA4PO algorithm can obtain better system response time and utilize fewer rules than Xu's DFS algorithm.

To fairly compare MOEA4PO and EA4PO algorithms, the setting of parameters is same in them. Similar to EA4PO algorithm, MOEA4PO algorithm was independently executed 20 times. And a set of average response time are computed with regard to different count of rule usage with improvement effect. They can be represented by different points and depicted in Figure 1. There are three kinds of different points. The first category, represented by circle points, is possibly excluded by DFS algorithm because of the confined search space. And it is difficult for EA4PO to find these solutions. The circle point indicates the response time of 22. 2ms obtained by MOEA4PO. It is better than the resulting response time obtained by EA4PO and DFS algorithms. The second category is composed of the two triangle points. The one point (26.32ms, 7) indicates that MOEA4PO is superior to EA4PO and DFS algorithms with regard to the response time and the count of rule usage with improvement. Another point (24.37ms, 8) shows that MOEA4PO is better than DFS. The rest points are square points. These solutions corresponding to square points can be used to compromise between the response time and the count of rule usage with improvement effect by software architect.

One of the primary reasons for such good results is that MOEA4PO algorithm always searches for a non-dominated set, rather than a single object which is combined response time with count of rules usage with improvement effect by means of weighing. As a result, MOEA4PO algorithm is more likely to explore widely in the search landscape. This also explains why MOEA4PO algorithm outperforms EA4PO algorithm.

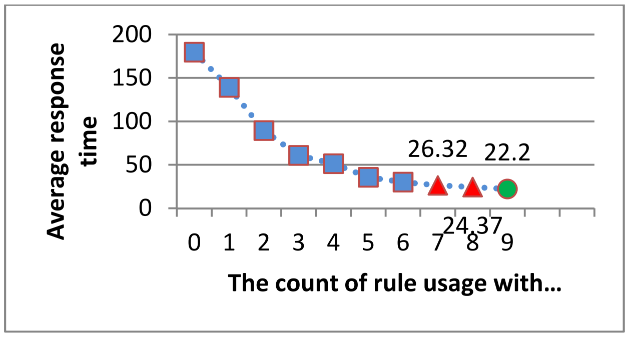

Figure 1. The average resulting response time and the count of rule usage with improvement effect obtained by MOEA4PO algorithm

5. CONCLUSION

The existing rule-based performance optimization approaches at software architecture level may only explore a limited region of the improvement space because they do not fully take the uncertainty of the count and order of each rule usage into account. Aiming at this problem, MOEA4PO algorithm is proposed to find the better solution by improving our previous EA4PO algorithm. The two factors of response time and the count of rules usage with improvement effect in EA4PO algorithm are considered as two objectives in MOEA4PO algorithm. Experimental results show that the more diverse solutions can be obtained in MOEA4PO by exploring the larger search space. Some solutions with better response time are difficult to be found by EA4PO algorithm. The other solutions indicate that MOEA4PO algorithm is superior to EA4PO and DFS algorithms with regard to the response time and the count of rule usage with improvement. These solutions obtained by MOEA4PO algorithm can support software architect to get trade-off candidates.

In the future, we will introduce cost function to evaluate the cumulative cost for the modification of *SA* caused by application of performance improvement solution. Furthermore, we will modify MOEA4PO algorithm by considering response time and cost as two objectives.

6. ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (No. 61305079, 61370078), the Natural Science of Fujian Province (No. 2015J01235), the open fund of State Key Laboratory of Software Engineering (No. SKLSE 2014-10-02).

7. References

[1] X. Du, Y. C. Ni, P. Ye, X. Yao, L. L. Minku, An Evolutionary Algorithm for Performance Optimization at Software Architecture. CEC, 2015.

[2] Xu, J., Rule-based automatic software performance diagnosis and improvement. Performance Evaluation, 2012, 69(11), pp. 525-550.